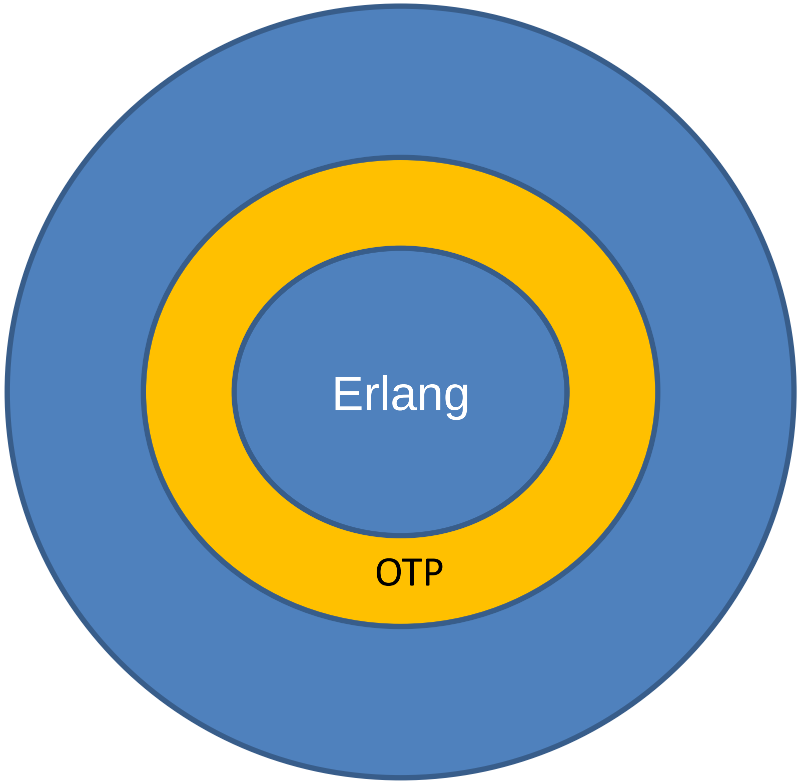
Cloud Scaling Erlang



Richard.Croucher@informatix-sol.com

www.informatix-sol.com

Platform Architect and Technologist

- Chief Architect at Sun Microsystems where I helped create the dotcom deployment standard and designed and deployed several large clusters with 200 – 1024 servers each
- Principle DevOPs Architect at Microsoft for all its Internet properties .
    - Adding 4000 servers a month. Established the dynamic computing working group to create design patterns for Cloud computing
- Primary focus over the last decade has been High Frequency Trading systems for Banks
    - pushing technology to extremes, shaving microseconds off distributed applications
- Involved with several Cloud startups
- Code in multiple languages
    - Transitioned through Assembler, Pascal, C, C++, Java , C# and Erlang

Degrees in Physics, Electronics and Materials Science from University of North London, Berkshire and Brunel
Fellow of the British Computer Society (FBCS)
Fellow of Securities and Technologies Council (STAC)
Charted IT Practitioner (CITP)

See www.informatix-sol.com Richard.Croucher@informatix-sol.com

My Erlang History

- Started with Erlang around 2011 after deduction that OO programming languages with their memory shared across all threads was increasingly a bad match to future hardware platforms
- Read the books and played around with Erlang, created a few programs, including my extracts db and mp3 collection management
- Commissioned to create a design for a new startup - Cloud service with multi factor communications. Requirement was for agile development and to scale to 1million users

My Erlang History

Informatix Solutions

- Started with Erlang around 2011 after deduction that OO programming languages with their memory shared across all threads was increasingly a bad match to future hardware platforms
- Read the books and played around with Erlang, created a few programs, including my extracts db and mp3 collection management
- Commissioned to create a design for a new startup - Cloud service with multi factor communications. Requirement was for agile development and to scale to 1million users
- Designed Erlang/OTP solution using RabbitMQ, Mnesia and Yaws as the main components
- Each User has dedicated Agent with runs continuously, pushing updates and handling requests from any of their registered devices (web browser, smartphone, pad)
- Message bus with Direct and Topic based Exchanges connects all major components
- Device specific gateways connect each type to the AMQP message bus
- Utilised many open source components including gen_smtp, jsx, larger, ejson, erlang-rfc4627, goldrush, mochiweb and webnesia
- Working with 3 part-time developers, created the functional system running on single nodes on AWS. Impressive how quickly such a small team could create such a powerful system
- Have been investigating approaches for how best to make it scale across multiple servers

The Erlang and OTP Advantage

- Erlang makes it easy to write the code that could scale across multiple servers
- Implicitly scales on SMP - leverage large servers to reduce overall count
- Actor based model and low cost processes
- Asynchronous Messaging
- Execute a function on a remote node, RPC
- Easy to create clusters
- `erl -sname hostname -setcookie mysecret`

The Erlang and OTP Advantage

- Erlang makes it easy to write the code that could scale across multiple servers
- Implicitly scales on SMP - leverage large servers to reduce overall count
- Actor based model and low cost processes
- Asynchronous Messaging
- Execute a function on a remote node, RPC
- Easy to create clusters
- erl -sname hostname -setcookie mysecret
- OTP provides many required services and generalised supervisor patterns which support the Erlang principle of crash early, automatically restart and enables reliable, distributed applications to be rapidly created
- Includes standard behaviours
- **gen_server**
- start_link(ServerName, Module, Args, Options)
- **gen_fsm**
- Finite State machine
- **gen_event**
- Event machine
- Supported by other components such as Mnesia (cluster database), Supervisors, Release control, Edoc, live updates, profiling, code inspectors etc.
- Hundreds of useful components on github

How many servers do we need to support 1 billion Users or IOTs?

Informatix Solutions

Assuming we a supporting these from a Erlang/OTP based platform running on higher performance Linux servers in an Agent based solution

- Let each User have an Agent, and let each Agent run 3 Erlang processes, therefore 3Billion Erlang processes required
- Assume each Erlang node can support 1 million Erlang processes → 3,000 servers

How many servers do we need to support 1 billion Users or IOTs?

Assuming we are supporting these from a Erlang/OTP based platform running on higher performance Linux servers in an Agent based solution

- Let each User have an Agent, and let each Agent run 3 Erlang processes, therefore 3Billion Erlang processes required
- Assume each Erlang node can support 1 million Erlang processes → 3,000 servers
- Assume active agent memory footprint is 2MB, idle Agent is 24KB and normal mix of 90% idle at any time, total memory requirement per node is:
- Active Agents footprint per node (each of 3000 servers) = 6.6GB
- Idle Agent footprint per node (each of 3000 servers) = 7.8GB
- ++ ETS, Message buffers..... → 32GB per server
- Erlang scales well on SMP so can use lots of cores, best price performance is 1 or 2 socket servers, which can currently provide up 80 cores

These are very rough estimates but are similar order of magnitude to the Cloud solutions supporting similar User populations

Distributed computing support in OTP?

OTP Distributed Application controller - dist_ac

- distributed = [{Application, [Timeout,] NodeDesc}]
- Where NodeDesc is a list of Nodes this application may execute on, in priority order
- For this to work, first the nodes must contact each other
- sync_nodes_mandatory = [Node] - all these nodes must be running
- sync_nodes_optional = [Node] - this nodes may be running
- Restarts application on the least loaded node, after 5secs, based on priority
- Design goal is to keep the application running in an environment where its assigned node(s) may fail
- Does not provide scalability and only very basic load balancing

Distributed computing support in OTP?

Informatix Solutions

OTP Distributed Application controller - dist_ac

- distributed = [{Application, [Timeout,] NodeDesc}]
- Where NodeDesc is a list of Nodes this application may execute on, in priority order
- For this to work, first the nodes must contact each other
- sync_nodes_mandatory = [Node]   - all these nodes must be running
- sync_nodes_optional = [Node] - this nodes may be running
- Restarts application on the least loaded node, after 5secs, based on priority
- Design goal is to keep the application running in an environment where its assigned node(s) may fail
- Does not provide scalability and only very basic load balancing

- Process Groups (pg2)
  - Manages groups of processes in a cluster with a common name
  - get_members, get_local_members – returns the PIDs, leaves it you to message them
  - Creates a monitor on each member to each of the rest of it’s group members
  - Breaks, unless you prevent multiple joins to the same group

Distributed computing support in OTP?

Informatix Solutions

OTP Distributed Application controller - dist_ac

- distributed = [{Application, [Timeout,] NodeDesc}]
- Where NodeDesc is a list of Nodes this application may execute on, in priority order
- For this to work, first the nodes must contact each other
- sync_nodes_mandatory = [Node] - all these nodes must be running
- sync_nodes_optional = [Node] - this nodes may be running
- Restarts application on the least loaded node, after 5secs, based on priority
- Design goal is to keep the application running in an environment where its assigned node(s) may fail
- Does not provide scalability and only very basic load balancing

- Process Groups (pg2)
  - Manages groups of processes in a cluster with a common name
  - get_members, get_local_members – returns the PIDs, leaves it you to message them
  - Creates a monitor on each member to each of the rest of it's group members
  - Breaks, unless you prevent multiple joins to the same group
- gen_server_cluster (Erlang Central)
  - enables multiple servers to provide a single service, each running on a different Erlang node. Implementation it to have one active server out of the cluster, sharing state to the others to they can elect a new leader if it goes down. Implementation does not provide scalability

Issues faced

- When creating a cluster you need to announce you’ve joined
  - net_adm:ping(some_other_node)

Issues faced

Informatix Solutions

- When creating a cluster you need to announce you've joined
  - net_adm:ping(some_other_node)
- No notification a node has left unless you setup monitors

Issues faced

- When creating a cluster you need to announce you’ve joined
  - net_adm:ping(some_other_node)
- No notification a node has left unless you setup monitors
- Placement is explicit e.g. spawn(**Node**, Module, Fun, [Args])

Issues faced

- When creating a cluster you need to announce you've joined
  - net_adm:ping(some_other_node)
- No notification a node has left unless you setup monitors
- Placement is explicit e.g. spawn(**Node**, Module, Fun, [Args])
- OTP pragmatically designed for the specialised compute environment of telephone switches, documented as being limited to 50-100 nodes

Issues faced

- When creating a cluster you need to announce you've joined
  - net_adm:ping(some_other_node)
- No notification a node has left unless you setup monitors
- Placement is explicit e.g. spawn(**Node**, Module, Fun, [Args])
- OTP pragmatically designed for the specialised compute environment of telephone switches, documented as being limited to 50-100 nodes
- OTP is focused on reliability, it's distribution components are to provide reliability not scalability

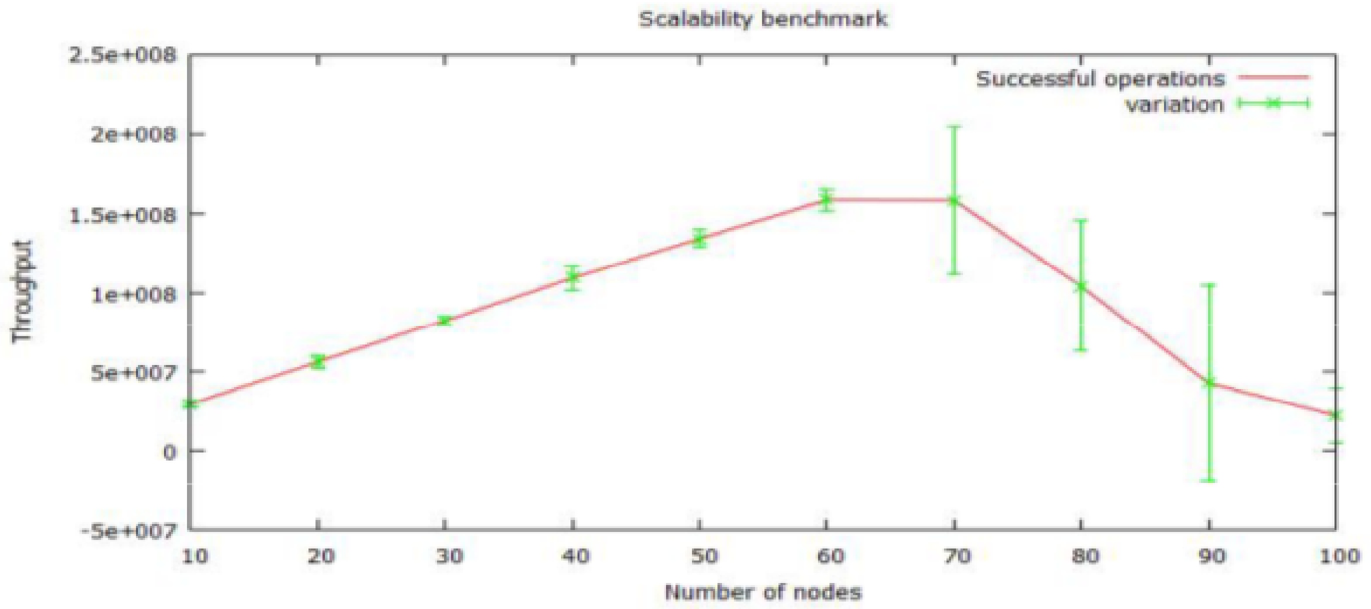
Issues faced

- When creating a cluster you need to announce you've joined
  - net_adm:ping(some_other_node)
- No notification a node has left unless you setup monitors
- Placement is explicit e.g. spawn(**Node**, Module, Fun, [Args])
- OTP pragmatically designed for the specialised compute environment of telephone switches, documented as being limited to 50-100 nodes
- OTP is focused on reliability, it's distribution components are to provide reliability not scalability
- OTP supervisors are constrained to only start processes on the same node. OTP recommendation is to use dist_ac

... surely someone has solved this?

Empirical Scalability results - Riak

Informatix Solutions



Benchmark on 348-node Kalkyl cluster at Uppsala University

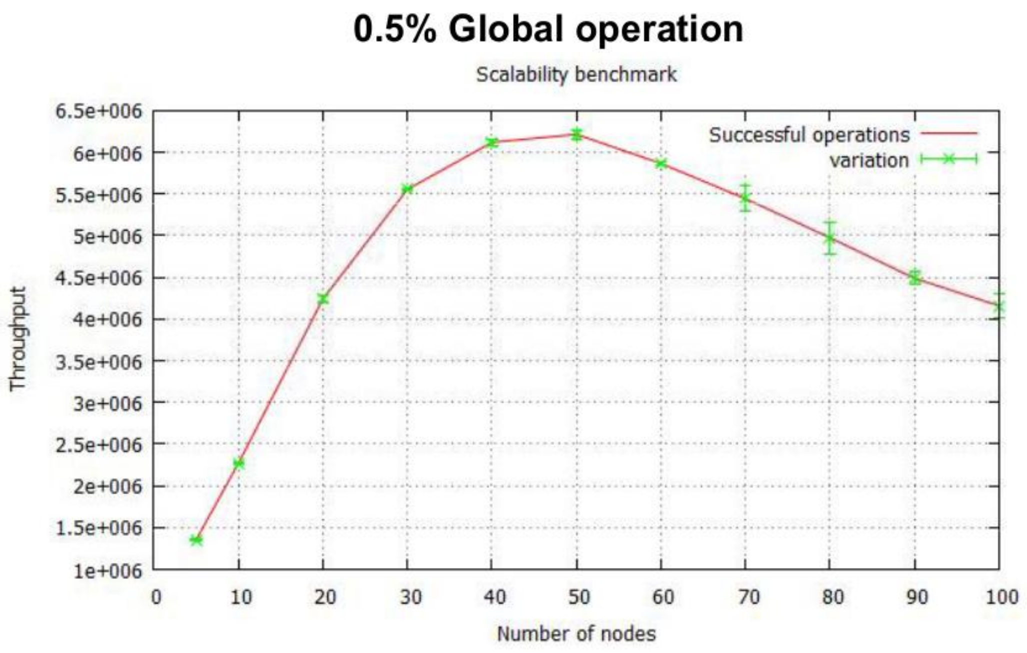
- Benchmark run as part of RELEASE program with Riak 1.1.1
- Tested with Basho Bench
- Initial investigation showed that Riak was scaling out at 60-70 nodes
- Investigation showed it was not constrained by physical resources -
  - Still CPU, memory or network B/W capacity available on each node

Recent improvements Riak are believed to improve this scalability

Scalable Persistent Storage for Erlang: Theory and Practice", Amir Ghaffari; Natalia Chechina; Phil Trinder, May 2013

Impact of global functions on scalability

Informatix Solutions



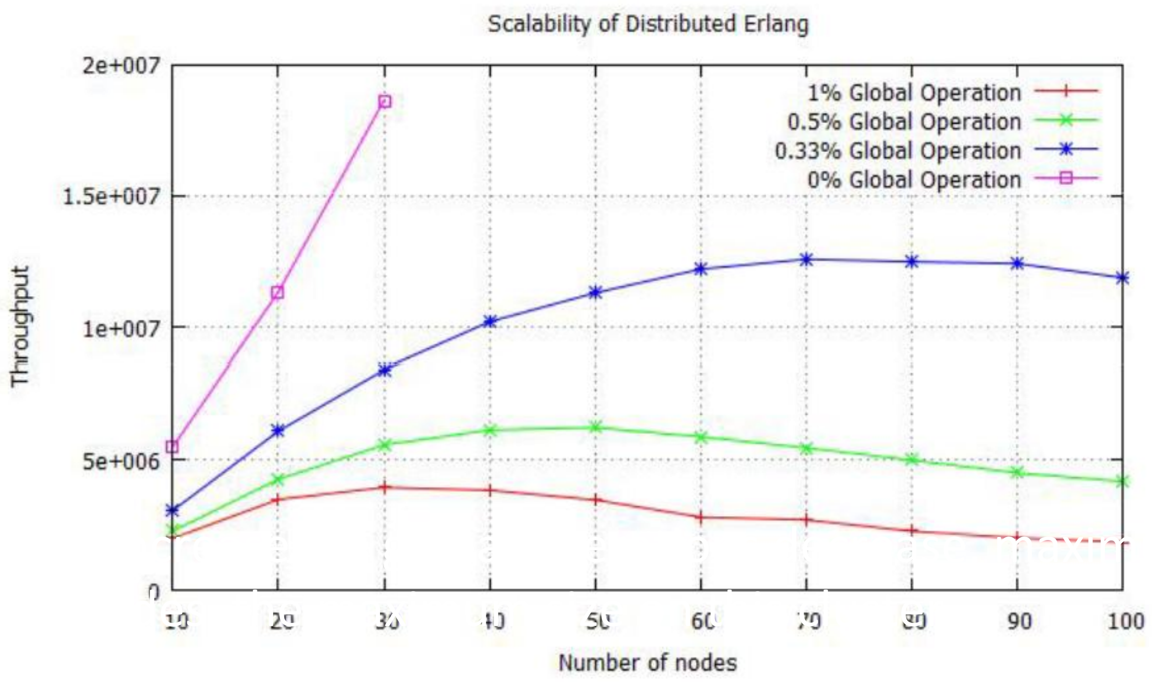
•Throughput peaks at 50 nodes
•Little improvement beyond 40 nodes

- Created DB-bench to enable experimental testing with different % of global operations
- Four commonly occurring functions in distributed programs:
  - whereis_name – lookup in a local table
  - spawn – peer to peer command
  - register_name – global name table, updated on every node
  - unregister_name – global name table, updated on every node
- Found that these were a close match to the observed Riak scalability

"Investigating the scalability limits of Distributed Erlang", Amir Ghaffari

Scalability variation with different % global operations

Informatix Solutions



Frequently	Max Throughput
1%	30 nodes
0.5%	50 nodes
0.33%	70 nodes
0%	1600 nodes

- Scalability inversely related to % of global operations
- Minimising these will improve scalability
- Hidden nodes help but impact manageability and flexibility
- Led to the creation of s_groups in SD Erlang

"Investigating the scalability limits of Distributed Erlang", Amir Ghaffari

Who's scaling today

Informatix Solutions

Few performance results published for configurations > 50 nodes but after some research and discussions

- Cloud providers:
  - WhatsApp - Chat/messaging, acquired by Facebook, 750m users, worlds largest messaging system
- Product Vendors:

WhatsApp - Probably biggest (single) Erlang system

Metrics

7m mesg/sec

Mnesia (Metadata store)

75K writes per sec per node
~36 billion records
~4TB RAM

Servers

> 1000 servers each with 20 cores, 64-512GB RAM
300 Chat servers
500 MMS Servers
FreeBSD 9.2, Erlang R16B
gen_server, gen_factory, pg2

Advice

Decouple, asynchronicity, partition

Who’s scaling today

Informatix Solutions

Few performance results published for configurations > 50 nodes but after some research and discussions

- Cloud providers:
  - WhatsApp - Chat/messaging, acquired by Facebook, 750m users, worlds largest messaging system
  - NASDAQ – live web media streaming, around 500 AWS nodes but tested to 1000 nodes
  - Oovuu -Cloud based video conferencing and 85 million Users, 5m concurrent, Riak storage layer, deploy as multiple scalable units, each with 6 RIAK servers, estimated at 20 servers per cluster
  - AOL Ad service – multiple clusters, each of up to 64 servers
  - Well known Ad Service – 79 node Riak cluster + 390 node Erlang cluster but don’t utilize any distributed services except for nodes().
  - Alert Logic - Cloud based alert and log file management, reportedly using a 1000+ Erlang nodes
  - Disco cluster – Nokia Research Pal Alto, 800 cores
- Product Vendors:
  - Basho Riak   - recommending caching with Redis or partitioning in multiple clusters above 100 servers
  - Couchbase - largest deployed cluster 80-100 nodes, tested up to 150 but discovered workload dependencies at around 120. Largest customer deployment over multiple clusters is 450+

Problems to solve building scalable solutions with Erlang and OTP

- Solve Explicit placement
  - No server is permanent in a large cluster
  - they fail, you add new ones, you remove small one to replace with bigger servers, you drop servers inline with demand, you take offline to upgrade

Problems to solve building scalable solutions with Erlang and OTP

Informatix Solutions

- Solve Explicit placement
  - No server is permanent in a large cluster
  - they fail, you add new ones, you remove small one to replace with bigger servers, you drop servers inline with demand, you take offline to upgrade
- Provide Load distribution and optimization- spawn new processes onto the optimal node
  - Find node with resources available e.g. Current load > 20% and < 80%
  - Add new servers to cluster when aggregate available spare capacity is below threshold
  - Remove server from cluster when current load < 5%

Problems to solve building scalable solutions with Erlang and OTP

- Solve Explicit placement
  - No server is permanent in a large cluster
  - they fail, you add new ones, you remove small one to replace with bigger servers, you drop servers inline with demand, you take offline to upgrade
- Provide Load distribution and optimization- spawn new processes onto the optimal node
  - Find node with resources available e.g. Current load > 20% and < 80%
  - Add new servers to cluster when aggregate available spare capacity is below threshold
  - Remove server from cluster when current load < 5%
- No Node discovery and announcement

Problems to solve building scalable solutions with Erlang and OTP

- Solve Explicit placement
  - No server is permanent in a large cluster
  - they fail, you add new ones, you remove small one to replace with bigger servers, you drop servers inline with demand, you take offline to upgrade
- Provide Load distribution and optimization- spawn new processes onto the optimal node
  - Find node with resources available e.g. Current load > 20% and < 80%
  - Add new servers to cluster when aggregate available spare capacity is below threshold
  - Remove server from cluster when current load < 5%
- No Node discovery and announcement
- Distributed Supervisor should be able to start and monitor process on different node (assuming a reliable network)

Problems to solve building scalable solutions with Erlang and OTP

- Solve Explicit placement
  - No server is permanent in a large cluster
  - they fail, you add new ones, you remove small one to replace with bigger servers, you drop servers inline with demand, you take offline to upgrade
- Provide Load distribution and optimization- spawn new processes onto the optimal node
  - Find node with resources available e.g. Current load > 20% and < 80%
  - Add new servers to cluster when aggregate available spare capacity is below threshold
  - Remove server from cluster when current load < 5%
- No Node discovery and announcement
- Distributed Supervisor should be able to start and monitor process on different node (assuming a reliable network)
- No Distributed Consensus and Master election – RAFT, PAXOS

Problems to solve building scalable solutions with Erlang and OTP

Informatix Solutions

- Solve Explicit placement
  - No server is permanent in a large cluster
  - they fail, you add new ones, you remove small one to replace with bigger servers, you drop servers inline with demand, you take offline to upgrade
- Provide Load distribution and optimization- spawn new processes onto the optimal node
  - Find node with resources available e.g. Current load > 20% and < 80%
  - Add new servers to cluster when aggregate available spare capacity is below threshold
  - Remove server from cluster when current load < 5%
- No Node discovery and announcement
- Distributed Supervisor should be able to start and monitor process on different node (assuming a reliable network)
- No Distributed Consensus and Master election – RAFT, PAXOS
- Reduce Chatiness of global functions

Problems to solve building scalable solutions with Erlang and OTP

Informatix Solutions

- Solve Explicit placement
  - No server is permanent in a large cluster
  - they fail, you add new ones, you remove small one to replace with bigger servers, you drop servers inline with demand, you take offline to upgrade
- Provide Load distribution and optimization- spawn new processes onto the optimal node
  - Find node with resources available e.g. Current load > 20% and < 80%
  - Add new servers to cluster when aggregate available spare capacity is below threshold
  - Remove server from cluster when current load < 5%
- No Node discovery and announcement
- Distributed Supervisor should be able to start and monitor process on different node (assuming a reliable network)
- No Distributed Consensus and Master election – RAFT, PAXOS
- Reduce Chatiness of global functions
- Need a Publish/Subscribe capability
  - Multi-casting in Erlang is Unicasting to each recipient

Some available Distributed Erlang components

Informatix Solutions

- SD Erlang (RELEASE program deliverables)
  - VM performance improvements that folded into current Erlang distro
  - s_groups to partition a cluster isolated in OTP branch.
  - Node_chooser and semi explicit placement
  - Various tools including benchErl , Dialyzer, Percept2

Some available Distributed Erlang components

Informatix Solutions

- SD Erlang (RELEASE program deliverables)
    - VM performance improvements that folded into current Erlang distro
    - s_groups to partition a cluster isolated in OTP branch.
    - Node_chooser and semi explicit placement
    - Various tools including benchErl , Dialyzer, Percept2
- Nodefinder
    - Uses multicast to discover other nodes in a cluster.
    - Separate version available to use on AWS (where multicast is prohibited)

Some available Distributed Erlang components

Informatix Solutions

- SD Erlang (RELEASE program deliverables)
  - VM performance improvements that folded into current Erlang distro
  - s_groups to partition a cluster isolated in OTP branch.
  - Node_chooser and semi explicit placement
  - Various tools including benchErl , Dialyzer, Percept2
- Nodefinder
  - Uses multicast to discover other nodes in a cluster.
  - Separate version available to use on AWS (where multicast is prohibited)
- Gproc
  - Global registry with state

Some available Distributed Erlang components

Informatix Solutions

- SD Erlang (RELEASE program deliverables)
  - VM performance improvements that folded into current Erlang distro
  - s_groups to partition a cluster isolated in OTP branch.
  - Node_chooser and semi explicit placement
  - Various tools including benchErl , Dialyzer, Percept2
- Nodefinder
  - Uses multicast to discover other nodes in a cluster.
  - Separate version available to use on AWS (where multicast is prohibited)
- Gproc
  - Global registry with state
- Riak_core
  - Node watcher to manage cluster membership, includes API to advertise and discover specific services. Enables adding and removing nodes
  - Master/worker pattern using vnodes as workers.
  - Stores cluster global state in the (Dynamo) ring by 'gossiping'
  - Which node to use depends on hashing within request ala Dynamo, filters out down nodes, relies in ability of any node to service request
  - Can be used generically but does have a strong Riak bias

Some available Distributed Erlang components

- SD Erlang (RELEASE program deliverables)
  - VM performance improvements that folded into current Erlang distro
  - s_groups to partition a cluster isolated in OTP branch.
  - Node_chooser and semi explicit placement
  - Various tools including benchErl , Dialyzer, Percept2
- Nodefinder
  - Uses multicast to discover other nodes in a cluster.
  - Separate version available to use on AWS (where multicast is prohibited)
- Gproc
  - Global registry with state
- Riak_core
  - Node watcher to manage cluster membership, includes API to advertise and discover specific services. Enables adding and removing nodes
  - Master/worker pattern using vnodes as workers.
  - Stores cluster global state in the (Dynamo) ring by 'gossiping'
  - Which node to use depends on hashing within request ala Dynamo, filters out down nodes, relies in ability of any node to service request
  - Can be used generically but does have a strong Riak bias
- RDMA Dist
  - James Lee prototype RDMA driver for Erlang. Sends messages over RDMA transport rather than TCP/IP

Some of the solutions available

Hundreds of components on github but at various stages of completeness and usability, all require integration and test

- Scalable Distributed (SD) Erlang (https://github.com/release-project/otp/tree/17.4-rebased)
- nodefinder (https://code.google.com/p/nodefinder/)
- Schemafinder (https://code.google.com/p/schemafinder/)
- RAFT consensus (http://raftconsensus.github.io/)
- riak_core (https://github.com/basho/riak_core)
- Computerl(https://github.com/paulgray/ComputErl)
- rdma_dist (https://github.com/MrStaticVoid/rdma_dist)
- Disco mapReduce cluster http://disco.readthedocs.org/en/latest/intro.html
- Gproc (https://github.com/uwiger/gproc)

- However, most big Cloud shops also talk about the changes they've been forced to make in low level components – I/O scheduler, gen_server, BEAM

this has suddenly become much more difficult

Networks and scalability

- Scalability is virtually always limited by network latency rather than B/W

Networks and scalability

- Scalability is usually limited by network latency rather than B/W
- Local networks , particularly L2, are more reliable than computers
- Modern networks are now faster than the computers we connect
- 10/40/56/100Gb NICs are now available
- Higher B/W also lowers latency due to the cost of packet serialization

Networks and scalability

- Scalability is usually limited by network latency rather than B/W
- Local networks , particularly L2, are more reliable than computers
- Modern networks are now faster than the computers we connect
  - 10/40/56/100Gb NICs are now available
  - Higher B/W also lowers latency due to the cost of packet serialization
- TCP/IP bottlenecks both throughput and latency as NICs > 10Gb are used

Networks and scalability

- Scalability is usually limited by network latency rather than B/W
- Local networks , particularly L2, are more reliable than computers
- Modern networks are now faster than the computers we connect
- 10/40/56/100Gb NICs are now available
- Higher B/W also lowers latency due to the cost of packet serialization
- TCP/IP bottlenecks both throughput and latency as NICs > 10Gb are used
- The highest performant compute clusters all rely on Remote Direct Memory Access (RDMA) to increase B/W, lower latency and reduce CPU overhead

Networks and scalability

- Scalability is usually limited by network latency rather than B/W
- Local networks , particularly L2, are more reliable than computers
- Modern networks are now faster than the computers we connect
- 10/40/56/100Gb NICs are now available
- Higher B/W also lowers latency due to the cost of packet serialization
- TCP/IP bottlenecks both throughput and latency as NICs > 10Gb are used
- The highest performant compute clusters all rely on Remote Direct Memory Access (RDMA) to increase B/W, lower latency and reduce CPU overhead
- The worlds largest stock exchanges all use RDMA to match orders in the fastest time, e.g. NASDAQ 5.5million msg/sec, others included NYSE, London Stock Exchange, Singapore
- The worlds largest RDBMS all turn to RDMA to maximize performance – Oracle, DB2, Microsoft SQLserver
- The worlds largest clustered file systems all use RDMA – Lustre, GPFS, DDN, Panansas

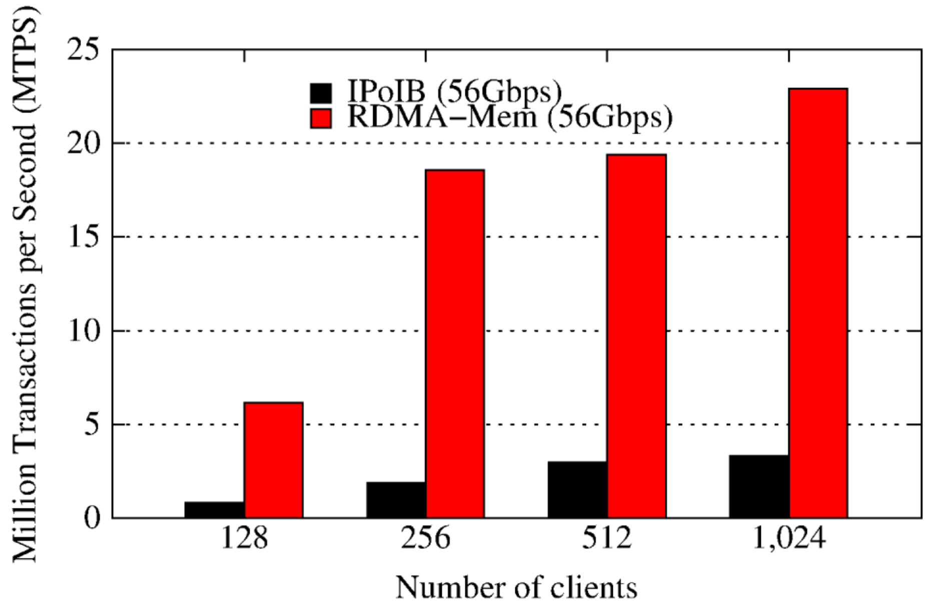
Networks and scalability

- Scalability is usually limited by network latency rather than B/W
- Local networks , particularly L2, are more reliable than computers
- Modern networks are now faster than the computers we connect
- 10/40/56/100Gb NICs are now available
- Higher B/W also lowers latency due to the cost of packet serialization
- TCP/IP bottlenecks both throughput and latency as NICs > 10Gb are used
- The highest performant compute clusters all rely on Remote Direct Memory Access (RDMA) to increase B/W, lower latency and reduce CPU overhead
- The worlds largest stock exchanges all use RDMA to match orders in the fastest time, e.g. NASDAQ 5.5million msg/sec, others included NYSE, London Stock Exchange, Singapore
- The worlds largest RDBMS all turn to RDMA to maximize performance – Oracle, DB2, Microsoft SQLserver
- The worlds largest clustered file systems all use RDMA – Lustre, GPFS, DDN, Panansas
- RDMA has been included in the Linux kernel since 2.6.14
- Transports over InfiniBand, RDMA enabled Ethernet cards, iWARP, Intel Omni-Path
- Latest 100Gb/s dual port NICs and switches achieving 195Gb/s, applications latency level of 610 nanoseconds and message rate of 149.5 million messages per second

http://www.hpcwire.com/off-the-wire/mellanox-connectx-4-100gbs-interconnect-adapter-delivers-record-performance-results/

- Intel adding RDMA directly to the CPU with their Omni-Path interconnect on ‘Skylake’

RDMA increases scalability for memcached



Memcached performance comparisons of TCP v. RDMA
36 node cluster, 32 requesters, 4 servers
Same physical 56Gb network interface used in both cases
Running RDMA enhanced memcached , in-memory mode, 4KB message size

Network-Based Computing Laboratory, Ohio State University
http://hibd.cse.ohio-state.edu/performance/scalability/

How to program with RDMA

Informatix Solutions

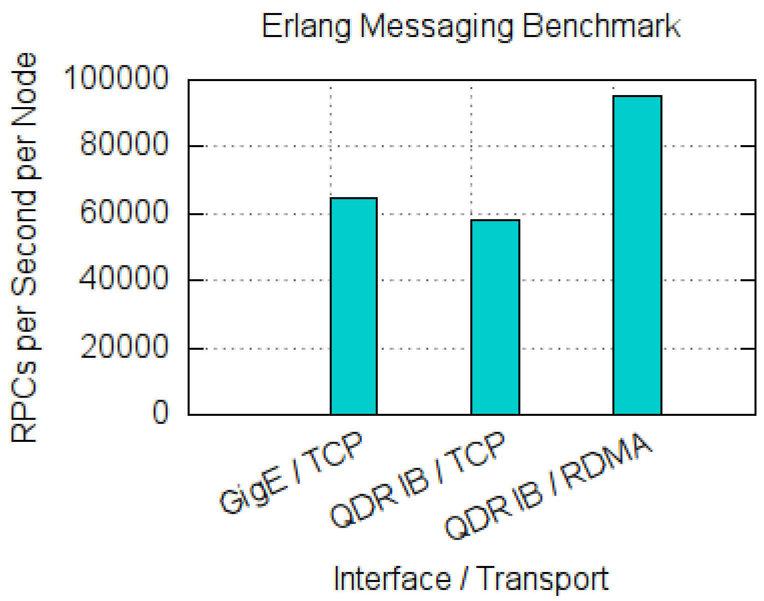
The simplest view of using RDMA is these steps:

1. register the memory you want to expose to RDMA
2. Establish a connection to the remote node - establishing a queue pair QP.
3. Send command to copy a block of memory (or pull a block memory). Can be upto 2GB or a linked list of buffers
4. Do something else whilst the RDMA hardware does all the work
5. When the response is received on the completion queue you will receive an interrupt to say it's finished

- It's a 'C' level API and you need to be able to manage where your data us going, which is much more difficult in languages like Java
- It’s a good fit into the asynchronous messaging behaviour of Erlang

RDMA with Erlang

Informatix Solutions



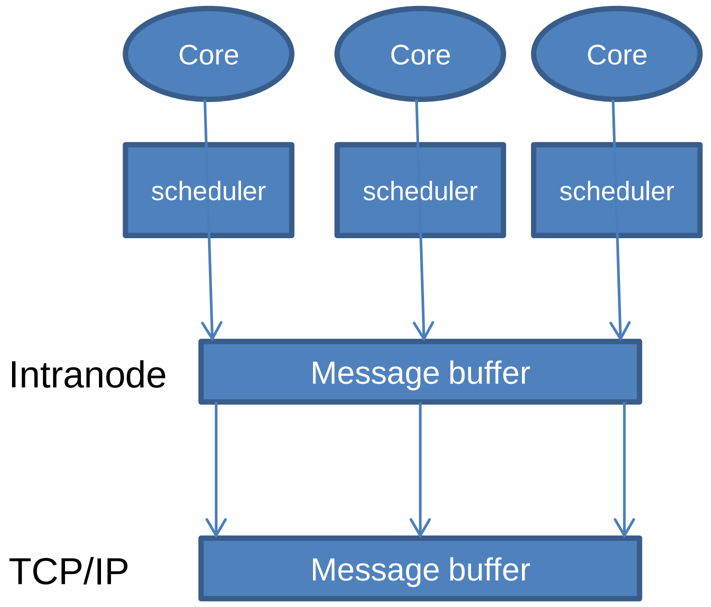
James Lee's very basic prototype RDMA Dist achieved 33% increase in message rate

Tuning and implementing directly into the message layer of the VM are expected to significantly increase this (as experienced in other environments)

Proposal for Erlang Message Queues to use RDMA

Informatix Solutions

- Intranode
  - Local messages are copied
  - Binaries >64 bytes are passed by reference

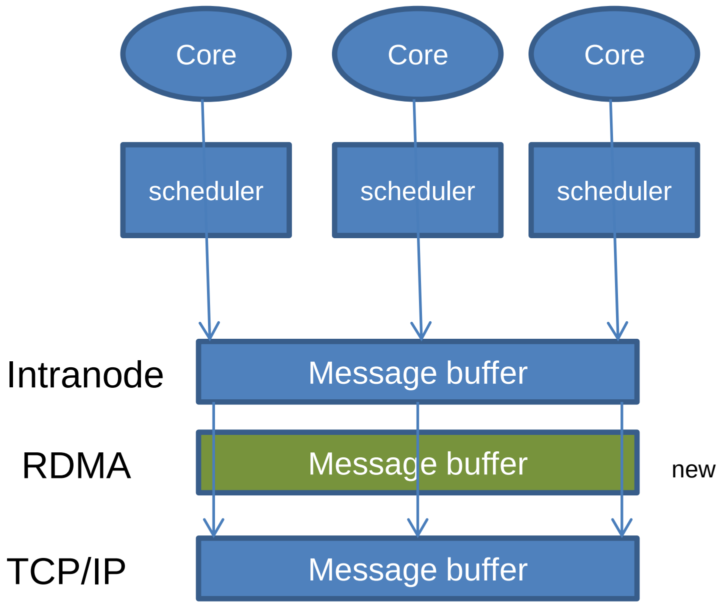


- TCP/IP
  - Message to remote transferred via TCP/IP connection
  - Encoded in Erlang External format to support heterogeneous environments

Proposal for Erlang Message Queues to use RDMA

Informatix Solutions

- Intranode (current behaviour)
  - Local messages are copied
  - Binaries >64 bytes are passed by reference
- RDMA (new)
  - Traffic to RDMA connected nodes passed using RDMA to replicate message between communicating nodes
  - Assumes homogenous cluster
  - Available for Linux, Windows, Solaris, AIX and more
- TCP/IP (current behaviour)
  - Message to remote transferred via TCP/IP connection
  - Encoded in Erlang External format to support heterogeneous environments

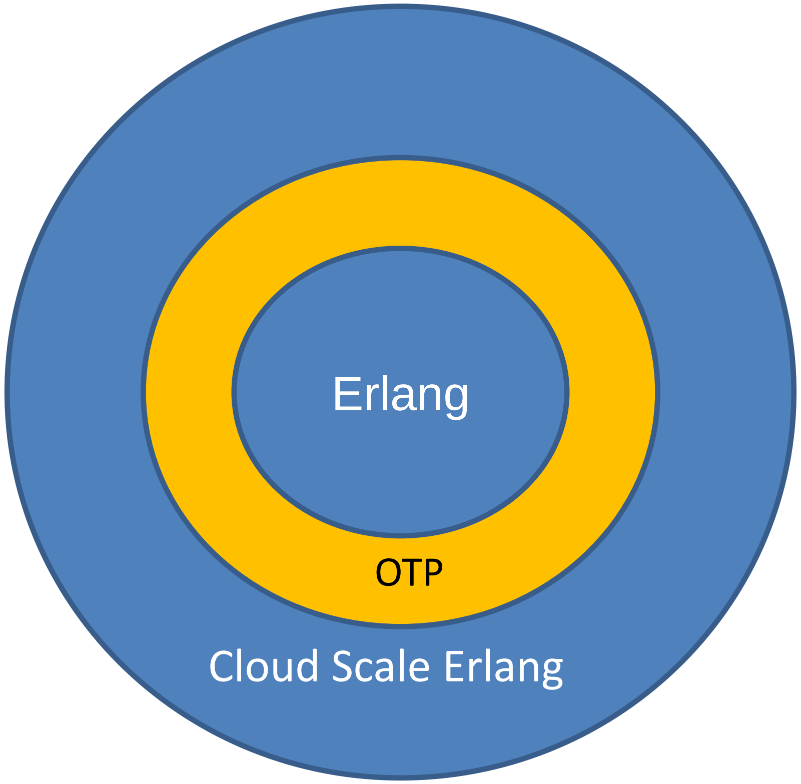


- RDMA works by registering area's of memory to be shared between nodes
- Blocks (up to 2GB) within this can then be written to a remote node or read from remote node at close to wire speed (40/56/100Gb/s) and with minimal CPU cycles
- Signalling via completion events manages synchronization and prevents accessing transitional memory values
- Atomic test and set VERBs available to enable distributed locking
- Enables Erlang to take advantage of the next generation of networking

Cloud Scale Erlang Platform?

Informatix Solutions

- Create a platform to extend Erlang/OTP into the super scaler space
- Integrate and test many of the existing components from github
- Add new capabilities, OTP behaviours servers and modules
- Encourage the Cloud community to provide their enhancements
- Preserves highly reliable Erlang /OTP core
- Work with Erlang/OTP team to put back scalability changes that do not conflict with OTP reliability goals
- Cloud Scale Erlang kept in step with OTP releases
- Make Erlang the premier platform for Cloud deployments



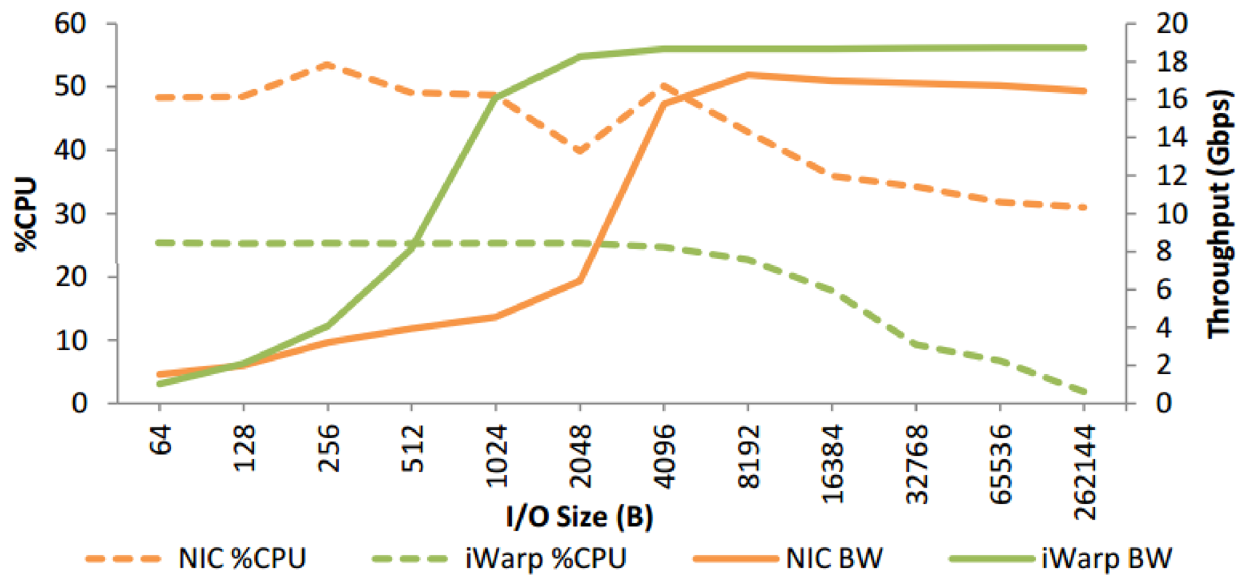
Who should do this?
Industrial Erlang User Group?
Create Apache Foundation project?

Richard.Croucher@informatix-sol.com

Slides will be here shortly - www.informatix-sol.com

SPARE SLIDES

RDMA increases throughput and lowers CPU utilization



- Shows difference in throughput and CPU utilization for different payload sizes transferred by RDMA and native Ethernet
- Tests were carried out using Chelsio 40G Ethernet cards which include iWARP support to run RDMA.
- Higher bandwidth and lower CPU utilization achieved with RDMA

Results courtesy of Chelsio

Scalable Distributed (SD) Erlang

- A EU funded program with collaborators including Uppsala University, Herriot Watt University, University of Kent, Erlang Solutions, Ericsson, University of Glasgow, EDF
- Goal was to evolve Erlang to run on 100K core environments
- Carried out projects to improve scalability of the Erlang VM, defined a distributed component Ontology, prototyped a cloud deployment tool (Wombat)
- Current (May 2015) software available:
  - Scalable Distributed Erlang – OTP branch
  - Tools – BenchErl, DEbench, Dialyzer, Percept2, ErLLVM, Concuerror, Devo, Wrangler, Orbit
- White papers talk about:
  - s_groups will allow partitioning of a cluster and reduce chatter.
  - Semi-explicit placement, replacing the default round-robin placement, placement hints to encourage deployment in same node for example.
  - load management - planned to add load server, will collect load information and decide where to spawn a new process. Will have one load server per node, choose_node function

http://www.release-project.eu/ https://github.com/release-project